#### **浅拷贝和深拷贝**

JavaScript有两种数据类型，基础数据类型和引用数据类型。基础数据类型都是按值访问的，我们可以直接操作保存在变量中的实际的值。而引用类型如Array，我们不能直接操作对象的堆内存空间。引用类型的值都是按引用访问的，即保存在变量对象中的一个地址，该地址与堆内存的实际值相关联。

****一、深拷贝和浅拷贝的区别****

  浅拷贝（shallow copy）：只复制指向某个对象的指针，而不复制对象本身，新旧对象共享一块内存；   
  深拷贝（deep copy）：复制并创建一个一摸一样的对象，不共享内存，修改新对象，旧对象保持不变。

var a = 25;var b = a;

b = 10;

console.log(a);//25

console.log(b);//10

//浅拷贝var obj1 = { a: 10, b: 20, c: 30 };var obj2 = obj1;

obj2.b = 40;

console.log(obj1);// { a: 10, b: 40, c: 30 }

console.log(obj2);// { a: 10, b: 40, c: 30 }

//深拷贝var obj1 = { a: 10, b: 20, c: 30 };var obj2 = { a: obj1.a, b: obj1.b, c: obj1.c };

obj2.b = 40;

console.log(obj1);// { a: 10, b: 20, c: 30 }

console.log(obj2);// { a: 10, b: 40, c: 30 }

****二、浅拷贝的实现****

var json1 = {"a":"name","arr1":[1,2,3]}function copy(obj1) {

　　　　var obj2 = {};

　　　　for (var i in obj1) {

　　　　　　obj2[i] = obj1[i];

　　　　}

　　　　return obj2;

}var json2 = copy(json1);

json1.arr1.push(4);

alert(json1.arr1); //1234

alert(json2.arr1) //1234

****三、深拷贝的实现****

****1、**Object.assign()**

let foo = {

a: 1,

b: 2,

c: {

d: 1,

}

}

let bar = {};

Object.assign(bar, foo);

foo.a++;

foo.a === 2 //true

bar.a === 1 //true

foo.c.d++;

foo.c.d === 2 //true

bar.c.d === 1 //false

bar.c.d === 2 //true

  Object.assign()是一种可以对非嵌套对象进行深拷贝的方法，如果对象中出现嵌套情况，那么其对被嵌套对象的行为就成了普通的浅拷贝。

****2、转成JSON****

  用JSON.stringify把对象转成字符串，再用JSON.parse把字符串转成新的对象。

var obj1 = { body: { a: 10 } };

var obj2 = JSON.parse(JSON.stringify(obj1));

obj2.body.a = 20;

console.log(obj1); // { body: { a: 10 } }

console.log(obj2); // { body: { a: 20 } }

console.log(obj1 === obj2); // false

console.log(obj1.body === obj2.body); // false

  但这种方法的缺陷是会破坏原型链，并且无法拷贝属性值为function的属性

****3、递归****

  采用递归的方法去复制拷贝对象

var json1={"name":"shauna","age":18,"arr1":[1,2,3,4,5],"string":'got7',"arr2":[1,2,3,4,5],"arr3":[{"name1":"shauna"},{"job":"web"}]};var json2={};function copy(obj1,obj2){

var obj2=obj2||{};

for(var name in obj1){

if(typeof obj1[name] === "object"){

obj2[name]= (obj1[name].constructor===Array)?[]:{};

copy(obj1[name],obj2[name]);

}else{

obj2[name]=obj1[name];

}

}

return obj2;

}

json2=copy(json1,json2)

json1.arr1.push(6);

alert(json1.arr1); //123456

alert(json2.arr1); //12345

#### **new 操作符**

一. new是干嘛的?

new操作符用来生成一个新的对象, 它后面必须跟上一个函数(否则, 会抛出TypeError异常), 这个

函数就是我们常说的构造函数.

二. new操作构造函数生成实例的过程

(1) 首先, 当我们使用new操作符时, js会先创建一个空的对象;

(2) 然后, 构造函数中的this指向该空对象;

(3) 其次, 在构造函数中通过操作this, 来给这个空对象赋予相应的属性;

(4) 最后, 返回这个经过处理的"空对象"(此时, 对象已经不是空的了).

****三.****new操作构造函数的注意事项

(1) 如果构造函数的返回值是一个原始类型(非引用对象, 如字符串), 那么返回值为new创建的"空对象", 如:

       var obj = new function() { return "code";};  
       alert(obj); // [object object]

(2) 如果构造函数的返回值是一个引用对象(数组, 对象, 函数等), 那么返回值会覆盖new创建的"空对象", 如:  
       var obj = new function() { return new String("code"); };  
       alert(obj); // "code"

<https://www.cnblogs.com/onepixel/p/5043523.html>

#### **原型是什么，原型的主要作用**



分析一下，上面这个例子！我们可以知道 People的类型是一个对象！  
按照javascript的说法，function定义的这个Person就是一个Object(对象),而且还是一个很特殊的对象，这个使用function定义的对象与使用new操作符生成的对象之间有一个重要的区别。这个区别就是function定义的对象有一个prototype属性，使用new生成的对象就没有这个prototype属性，我们一般称为普通对象！  
我们需要理解记忆以下的逻辑顺序：  
Person是一个对象，它有一个prototype的原型属性（因为所有的对象都一prototype原型！）prototype属性有自己的prototype对象，而pototype对象肯定也有自己的constuct属性，construct属性有自己的constuctor对象，神奇的事情要发生了，这最后一个constructor对象就是我们构造出来的function函数本身！

二、这样绕人的逻辑，总想一探究竟，要记得这是一种规定的逻辑思维，我们要做的是慢慢接受这种逻辑思维！我们不理论研究，用实例说话：



三、下面我们不免有了疑问，学这么绕的东西干什么，它给我们最实际的用处就是我们可以用原型来创建对象的属性和方法！我们不用它不也是可以创建属性和方法！这里是有区别的，既然不一样就有存在的价值！  
     我们可以通过给原型添加属性和方法来给给对象添加属性或方法！  
    Hero.prototype.name;  
    Hero.prototype.sayMe = function(){"添加对象的方法其实就是添加函数"}  
  
    让我们再深一步：当我们给对象添加了同名的属性或方法时会发生什么？



我们验证了这个例子得到得到了一些结论：  
当函数对象本身的属性或方法与原型的属性或方法同名的时候：  
    1、默认调用的是函数对象本身的属性或方法.  
    2、通过原型增加的属性或方法的确是存在的.  
    3、函数对象本身的属性或方法的优先级要高于原型的属性或方法.

#### **清除浮动的几种方法**

如何脱离标准文档流，css中给了三种方法，使一个元素脱离标准文档流。

   1.浮动。   2.绝对定位。  3.固定定位。

一、 什么是浮动呢？定义这样说：浮动的框可以向左或向右移动，直到它的外边缘碰到**包含框**或**另一个浮动框的边框**为止。 其实也就是把快级元素变成能和别的快级并排成一行。

二、浮动的的元素有4点特性：

    1.浮动元素会脱离标准文档流，已经不区分快和行了。

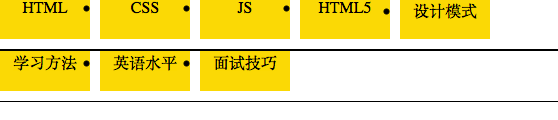
    2.浮动的元素会互相贴靠。

    3.浮动的元素有“字围”的效果。

    4.收缩。一个浮动的元素，如果没有设置width，那么将自动收缩为文字的宽度。

三、清除浮动的方法。

   1.给浮动的元素的祖先元素加高度（父元素或祖元素）。只要浮动在一个有高度的盒子中，那么这个浮动就不会影响后面的元素了，也就是说，如果盒子内部有浮动，这个盒子有高，那么妥妥的，浮动不会互相影响。所以也就清除了浮动带来的影响。浮动的元素，只能被有高度的盒子关住。但是，工作上，我们绝对不会给所有的盒子加高度，这是因为麻烦，并且不能适应页面的快速变化。（不建议使用）

    .

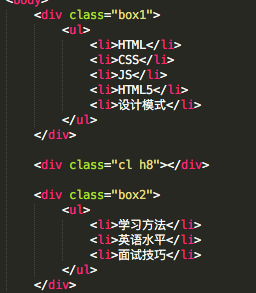
   2.clear：both；清除掉两边的浮动。但是有一个非常致命的问题，margin失效了，上下之间没有了间距。



3.隔墙法。就是在两个部分之间加一个<div>，设置一个高度。还有就是“内墙发”：就是在块里边在放一个<div>，设置高度，不仅不让后边的去追前边的，并且能给外边的<div>撑出一个高度。

  注：脱标的元素是不能把父元素撑出高度滴。

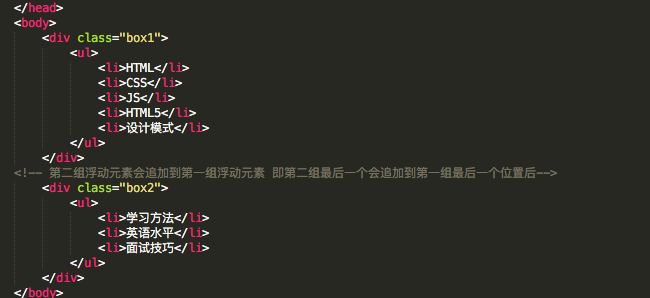
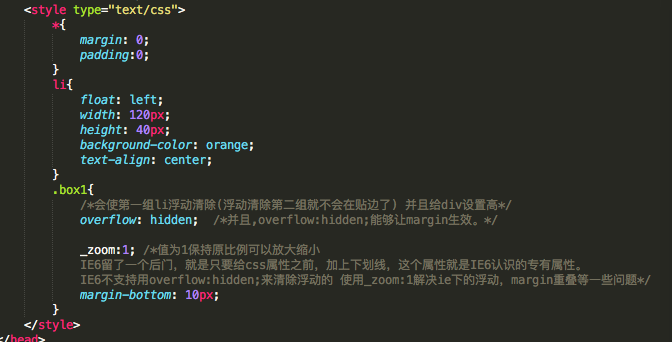
                外墙法：

       内墙法：



   4.overflow：hidden；  一个父元素不能被自己浮动的儿子撑出高度。但是，只要给父亲加上 overflow：hidden；那么父元素就能被儿子撑出高度了。  最最最重要的是，使用overflow：hidden；能使margin生效！你没有看错，就是酱。

#### **h5和css3的新特性**

#### **前端优化方法**

#### **兼容性有几点体现**

#### **判断对象的数据类型的几种方法**

#### **JS复杂数据类型**

#### **微信小程序、微信浏览器等**

#### **HTTP响应码**

#### **版本管理工具**

#### **打包工具**

#### **Call和apply的区别**

<http://www.cnblogs.com/onepixel/p/5143863.html>